

AD-A245 853



NAVSWC TR 90-46

2

EXPERT DESIGN ADVISOR

BY STEVEN L. HOWELL, PHILLIP Q. HWANG, AND CUONG M. NGUYEN
UNDERWATER SYSTEMS DEPARTMENT

OCTOBER 1990

Approved for public release; distribution is unlimited.

DTIC
ELECTE
FEB 10 1992
S B D



NAVAL SURFACE WARFARE CENTER

Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

92 2 06 002

92 2

102

92-03016



NAVSWC TR 90-46

EXPERT DESIGN ADVISOR

**BY STEVEN L. HOWELL, PHILLIP Q. HWANG, AND CUONG M. NGUYEN
UNDERWATER SYSTEMS DEPARTMENT**

OCTOBER 1990

Approved for public release; distribution is unlimited.

NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

CONTENTS

<u>Chapter</u>		<u>Page</u>
1	INTRODUCTION	1-1
	1.1 PROBLEM/SOLUTION	1-1
	1.2 BACKGROUND	1-3
2	MISSION CRITICAL SOFTWARE INTENSIVE SYSTEMS	2-1
	2.1 REQUIREMENTS DESCRIPTIONS	2-1
	2.1.1 FUNCTIONAL REQUIREMENTS	2-1
	2.1.2 NONFUNCTIONAL REQUIREMENTS	2-1
	2.1.3 BACKGROUND INFORMATION	2-2
	2.2 DESIGN DESCRIPTIONS	2-2
	2.2.1 LOGICAL MODEL	2-2
	2.2.2 IMPLEMENTATION MODEL	2-3
	2.3 IMPLEMENTATIONS.....	2-3
3	EDA ENHANCED SYSTEMS ENGINEERING METHODOLOGY	3-1
	3.1 STANDARD SOFTWARE INTENSIVE SYSTEMS	
	LIFE CYCLE MODEL	3-1
	3.1.1 PROBLEM DEFINITION	3-2
	3.1.2 SOFTWARE/SYSTEM DEVELOPMENT	3-3
	3.1.3 SOFTWARE SYSTEM MAINTENANCE (CHANGE CONTROL).....	3-5
	3.1.4 SUPPORT ACTIVITIES	3-5
	3.2 EXPERT DESIGN ADVISOR METHODOLOGY	3-6
	3.2.1 SYSTEM CHARACTERIZATION	3-7
	3.2.2 FRAGMENTATION AND DECOMPOSITION	3-9
	3.2.3 RECOMPOSITION	3-10
	3.2.4 RESOURCE ALLOCATION MAPPINGS	3-11
	3.2.5 CODE GENERATION	3-14
	3.2.6 AUTOMATION OPPORTUNITY OF METHODOLOGY	3-14
4	PROTOTYPE TOOLSET IMPLEMENTATION	4-1
	4.1 RESOURCE ALLOCATION ADVISOR TOOL	4-1
	4.1.1 SYSTEM REPRESENTATION	4-2
	4.1.2 FUNCTIONALITY.....	4-2
	4.1.3 DISPLAYS and COMMANDS.....	4-3
	4.2 EDA ENVIRONMENT	4-12
	4.2.1 SYMBOLICS AND KNOWLEDGE ENGINEERING ENVIRONMENT..	4-13
	4.2.2 VAX AND ADA.....	4-13
	4.3 EDA CAPABILITY	4-13
	4.3.1 ALLOCATIONS OF SOFTWARE MODULES ONTO HARDWARE	
	RESOURCES	4-13
	4.3.2 SIZE AND PERFORMANCE	4-13

FOREWORD

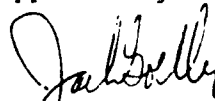
The Expert Design Advisor describes a methodology for the development of large, complex systems, such as Anti-Submarine Warfare systems, involving next generation mission critical computing resources. This work also demonstrates that the method is automatable using an artificial intelligence, expert system shell. The methodology of this system optimizes the design structure and generates near-optimal allocations of logical design objects onto physical implementations based on heuristic classes of rules, classes of algorithms, sets of analytical equations, and sets of probabilistic reasoning. This work should be of interest to systems engineers and systems analysts.

This work, performed at Naval Surface Warfare Center (NAVSWC), documents the state of the methodology as of FY90. The methodology is continuing to be defined and enhanced; further versions will be released in the future as part of the Engineering of Complex Systems Technology Block.

The project is sponsored by the Office of Naval Technology (ONT) and NAVSWC's Independent Exploratory Development program. It is a cooperative effort among NAVSWC, DoD, university and industry communities. Funding is expected to continue by ONT; this support will allow for the expansion of the existing effort.

The authors thank CDR Jane Van Fossen (ONT) and Dr. Frankie Moore for their programmatic support; Teresa Park and university and high school students Brian Crilly, Fredrick Mayus, and Dan Cornfeld for their technical support; and Adrien Meskin of Advanced Technology and Research Corporation for support in preparing this manuscript.

Approved by:



JACK GOELLER, Deputy
Underwater Systems Department

CONTENTS (Cont.)

<u>Chapter</u>		<u>Page</u>
	4.4 EDA USAGE	4-14
	4.4.1 RAA PROCESS	4-14
5	FUTURE PLANS AND DEVELOPMENT	5-1
	5.1 METHODOLOGY ENHANCEMENTS AND DEVELOPMENTS	5-1
	5.2 FUTURE VERSION OF TOOLSET	5-1
	5.2.1 CONTINUATION OF WORK	5-1
	BIBLIOGRAPHY	6-1
	NOMENCLATURE	7-1
	APPENDIX--SAMPLE ALGORITHM	A-1
	DISTRIBUTION	(1)

ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1-1	PROBLEM STATEMENT	1-2
1-2	EDA ARCHITECTURE	1-3
3-1	STANDARD SOFTWARE INTENSIVE SYSTEMS LIFE CYCLE PROCESS	3-1
3-2	EXAMPLES OF TWO VIEWS THAT DESCRIBE A SIMPLE SYSTEM	3-7
3-3	RECOMPOSITION OF A GRAPH	3-10
4-1	SYSTEM CHARACTERIZATION PANEL	4-4
4-2	ALGORITHM APPLIER PANEL	4-8
4-3	CANDIDATE ASSIGNMENT DISPLAY PANEL	4-11
4-4	EDA NETWORK	4-12
4-5	RAA TOOL FUNCTIONALITY OVERVIEW	4-15



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

CHAPTER 1

INTRODUCTION

The Expert Design Advisor (EDA) is a decision-aided toolset for use by systems engineers in facilitating the development of large, complex systems involving Mission Critical Computing Resources (MCCR).

1.1 PROBLEM/SOLUTION

EDA is specifically designed to avoid major errors and imperfections early in the design phase. In large, complex systems, involving MCCR, decisions impacting architectural development and performance cannot effectively be made by conventional inspection. When a poor design decision is made early in the design phase, the cost for correcting it may far exceed the planned budget. This is because all subsequent designs and implementations based on or derived from the poor decision may have to be scrapped and new plans started from scratch. This can be fatal for a particular project.

In a typical system, where EDA might be applied, there are at least three essential elements: requirements descriptions, design descriptions, and implementations. The requirements descriptions include measures of effectiveness (MOE) and environmental parameters; design descriptions include logical models (made up of many smaller logical modules) and implementation models (made up of many smaller implementation nodes); and implementations include hardware, software, manuals, and defined user interactions. The implemented design must satisfy the original requirements.

One function of the EDA is to map logical descriptions onto implementation resources in a distributed processing environment in order to optimize the performance of the overall system. From the "black box" point of view, as shown in Figure 1-1, the EDA prototype toolset utilizes the following inputs: implementation resources, logical descriptions, environmental parameters, MOE requirements, and user modeling information. It processes the input and produces candidate allocation mappings that provide high performance, high reliability, fault tolerance, and cost-efficiency to the overall system.

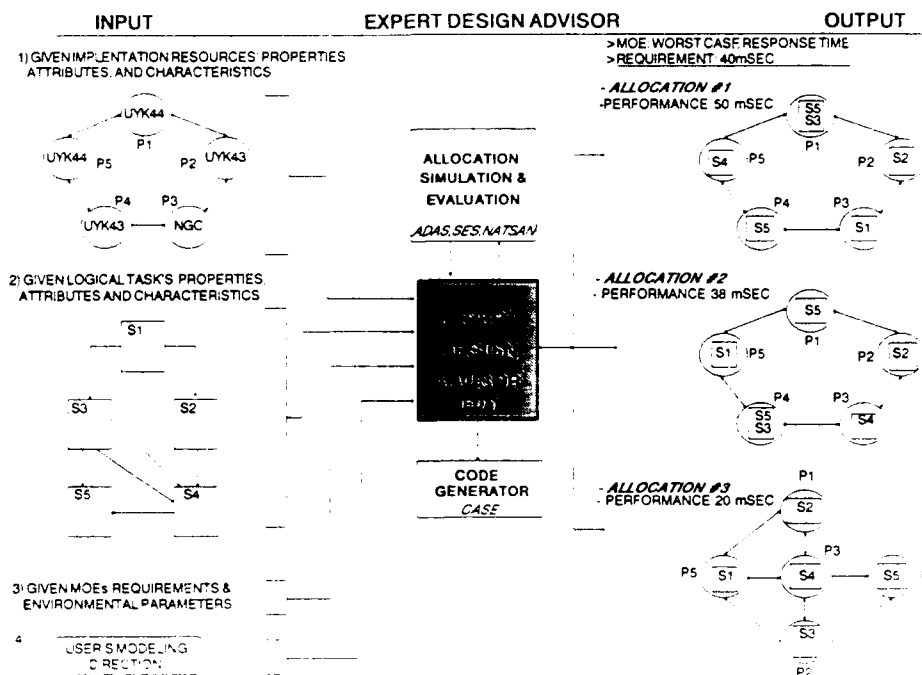


FIGURE 1-1. PROBLEM STATEMENT

The EDA Enhanced Systems Engineering Methodology is the standard software intensive systems life cycle model augmented by the EDA methodology that enhances all phases of the standard model. The EDA methodology is automated through the EDA prototype toolset, different types of simulation tools, different types of CASE tools, and the user interface as shown in Figure 1-2. The major development, however, is currently concentrated on the EDA prototype toolset since the other internal areas (i.e., CASE and simulation tools) have been addressed by commercial and DoD organizations.

The toolset (Figure 1-2) consists of five tools: the Recomposition Advisor (RA) tool, the Fragmentation and Decomposition Advisor (FDA) tool, the Code Generation Advisor (CGA) tool, the Utility tool, and the Resource Allocation Advisor (RAA) tool, the latter being the only tool that has been implemented to date.

The RA and FDA tools allow for the restructuring of logical or implementation models including the fragmentation, decomposition, and recomposition of components. The CGA tool analyzes the logical and implementation models and advises the Code Generator and the user on the structure of programs to generate. The Utility tool is a collection of routines that is used by the EDA prototype toolset to manipulate and transport the inputs, outputs, and system requirements. Lastly, the RAA tool analyzes both the logical and implementation models' characteristics and proposes an allocation mapping that satisfies the original requirements.

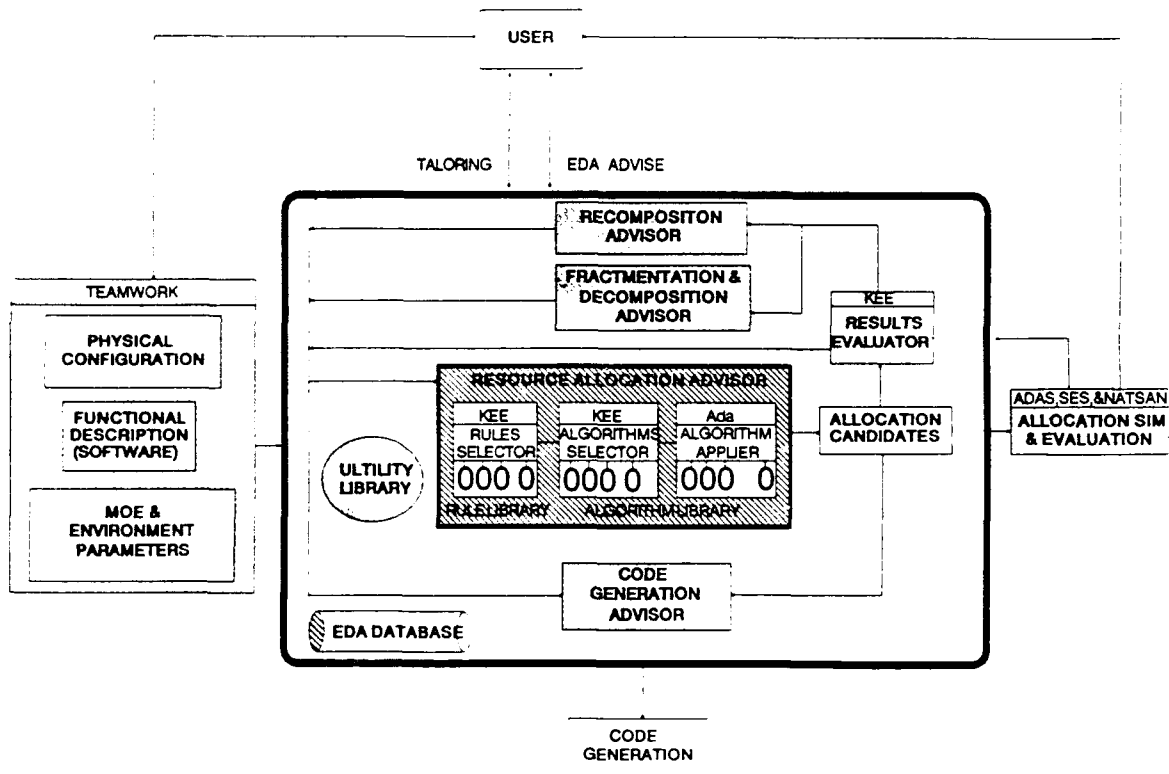


FIGURE 1-2. EDA ARCHITECTURE

The tools work independently, but complement each other in order to fulfill the functionality of the EDA. Given a particular view of the logical model and implementation model, the EDA prototype toolset attempts to make a near-optimal allocation.

1.2 BACKGROUND

The EDA was developed as an outgrowth of a larger Independent Exploratory Development (IED) research project. The objective of that effort was to develop an automated, integrated design methodology and design environment for the next generation Anti-Submarine Warfare (ASW) combat systems. The research was divided into two phases: (1) to establish, validate, verify, and document a system's design methodology for the implementation of Artificial Intelligence (AI) and ASW algorithms in VHSIC/VLSI technology, and (2) to automate that methodology to the maximum extent possible by developing an integrated expert system, data base, and design automation tools environment. The latter phase, which was based on the methods, techniques, and experience gained in the first phase, produced the EDA.

CHAPTER 2

MISSION CRITICAL SOFTWARE INTENSIVE SYSTEMS

Mission critical software intensive systems are defined by the various descriptions (or views) developed during the systems' life cycle, including its implementation. These descriptions specify the requirements, designs, and implementations of the system.

2.1 REQUIREMENTS DESCRIPTIONS

Requirements descriptions fall into three main categories: functional requirements, nonfunctional requirements, and background information.

2.1.1 Functional Requirements

The functional requirements describe what the system must do without specifying physical or temporal limitations. An example of a requirement for a sonar system is the following: The system must detect an object within a given range of the sonar. A more mathematical example is: The system must have the ability to sort numbers in ascending order.

2.1.2 Nonfunctional Requirements

Nonfunctional requirements specify or limit temporal and other requirements not directly related to tasks the system must perform. Examples of nonfunctional requirements include real-time constraints, reliability specifications, system size limitations, system production cost limitations, and environmental requirements (ruggedized/militarized).

2.1.2.1 Measures of Effectiveness (MOE). In order to determine how effectively the system meets its nonfunctional requirements, MOE are typically derived from the requirements and other programmatic sources. These MOE include, but are not limited to: worst case, average case, and/or best case response times for tasks/functions; response times for critical groups of tasks/functions; statistical distributions on response times; performance under load conditions; degradation under load; reliability (both functional and resource) measures; resource load constraints (functions, CPU utilization, memory utilization, disk utilization); resource size limitations; and resource cost limitations.

2.1.2.2 Environmental Parameters. The environmental parameters are variables outside of the system that might affect its operation as related to temporal and other nonfunctional requirements. They can be used to define load conditions. For example, if an ASW system is tracking the number of submarine contacts in an area, then an environmental parameter might be a worst case approximation of the number of submarines that might be detected at one time. A more mathematical example is a sort routine. If the sort routine is viewed as the system and it allows the user to specify how many numbers to sort, then the number of numbers to be sorted could directly affect the operation (response time) of the sort routine and is considered an environmental parameter.

2.1.3 Background Information

Included in the requirements is background information that does not define specific functional or nonfunctional requirements, but helps the systems engineer or systems analyst to better understand the reasons for building a particular system.

2.2 DESIGN DESCRIPTIONS

Design descriptions are made up of the logical and implementation model in addition to any supporting tools, documentation, and methods.

2.2.1 Logical Model

The logical model is typically designed in a hierarchical structure to encapsulate all the properties, attributes, and characteristics of the system's logical design. This hierarchical structure allows the logical model to be expanded, contracted, decomposed, and recombined without any limitation as the model evolves. By having these capabilities, the logical model framework is highly robust and highly flexible.

The design of the logical model assumes that resources are limitless and have all needed capability, but has no implementation information. During logical design, the multiple representation of the design is developed. Three typical representations include data design, behavioral design, and object-oriented design.

2.2.1.1 Data Design. In data design, the designer determines the elemental and composite types of data. These data types are influenced by the types of operations to be performed on them and will, in turn, affect the procedural design and interfaces. At this time, only abstract data types are considered; implementation details are left for later. How the data design affects subsequent design is determined by the methodology used: data flow oriented design leads to modular, sequential processing of relatively nonhierarchical data; data-structure oriented design leads to procedural processing of highly hierarchical data. The data design leads directly to architectural design.

2.2.1.2 Behavioral Design. In behavioral design, information and control are defined through use of modules and interfaces. At the logical model stage, the designer concentrates on design abstraction, modularity, and information hiding. Information concerning required abstract resources is annotated to some extent in the logical model.

2.2.1.3 Object-Oriented. The object-oriented descriptions define the objects that make up a system and the relationships between objects. In addition to the relationships between objects, capabilities of the objects are defined.

2.2.2 Implementation Model

While the logical model is being developed, physical implementation factors are also considered. Candidate implementations, consisting of limited resources of known functionality, are suggested for allocation to logical resources in the logical model. The breakdown and creation of physical implementation elements often reflect the possible programming languages and hardware capability for the system so that these elements can be represented.

Like the logical model, the implementation model is designed in a hierarchical structure to encapsulate all the physical properties, attributes and characteristics of the implementation model. It can also capture the relationship and the connectivity of the nodes, subnodes, supernodes, and network topology. This hierarchical structure allows the implementation network (or physical network) to be expanded and contracted without any bound limitation as the network evolves. By having these capabilities, the implementation model framework is highly robust and highly flexible.

2.3 IMPLEMENTATIONS

Implementations include all hardware devices as well as all software code, execution environment, data files, system-user documentation, and human interaction necessary for the system to perform its desired functions. Hardware may include computers, signal processors, sensors, and weapons. The software not only includes the binary code running on the computers, but also the source code with comments. The documentation may include information concerning design and/or requirements of the system as well as the procedures for using the systems. Part of the functionality that the system performs may in actuality be performed by the operator. In this case the operator is considered part of the system.

CHAPTER 3

EDA ENHANCED SYSTEMS ENGINEERING METHODOLOGY

The EDA Enhanced Systems Engineering Methodology is the standard software intensive systems life cycle model (i.e., "Waterfall Model," "Modified Waterfall Model," and "Spiral Model") augmented by the EDA methodology. This enhanced methodology attempts to improve all phases of the standard life cycle model and facilitate the development of systems that meet their requirements, are developed within cost, and are maintainable. The methodology is repeatable and, to a large extent, automated.

3.1 STANDARD SOFTWARE INTENSIVE SYSTEMS LIFE CYCLE MODEL

The software intensive systems life cycle model is a process by which a system is developed and maintained. It comprises three phases: problem definition, software/system development, and software/system maintenance. Figure 3-1 describes the components of the development process and their relationships to each other.

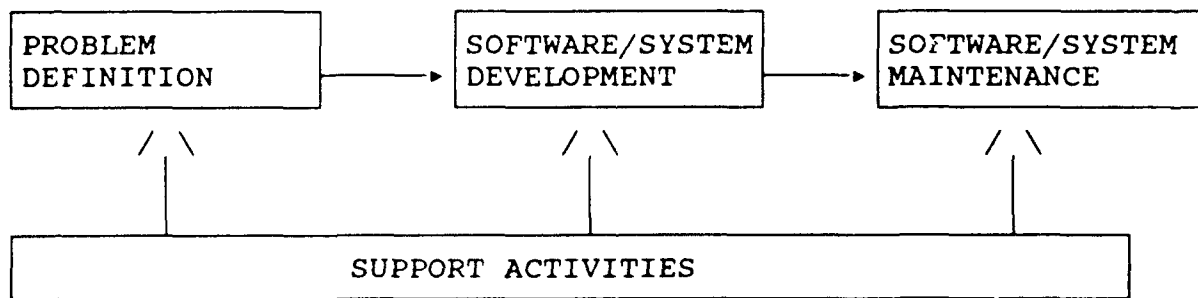


FIGURE 3-1. STANDARD SOFTWARE INTENSIVE SYSTEMS
LIFE CYCLE PROCESS

The three phases attempt to logically move the developers from the problem to the solution. This is sometimes called "the Waterfall Model" approach because each phase (and each subphase within a phase) follows one after another. However, in reality, it is often necessary for the developers to iterate back to a previous phase. There are two basic instances when this iteration will occur: when an error is discovered in a phase (after the phase is complete) and when the requirements and/or problem definition is changed due to a changing environment. In each of these instances the developer will

go back to the phase (and subphase) where the "problem" has occurred and make the necessary modifications to the phase (and subphase) and to all successive phases (and subphases) until they get back to the current phase (and subphase). This method is time consuming and cost inefficient. Other methods, such as the "Spiral Model," can also be used with the EDA methodology.

In addition to these three phases, there are many support activities that must take place for a software project to produce and maintain a usable solution. These include configuration management, project management, and document generation activities.

3.1.1 Problem Definition

The problem definition phase attempts to produce a clear understanding of the problem. The result of the phase is usually a document(s) that describes the problem in detail. A determination is also made by the end of the problem definition phase as to whether a problem warrants the development of the system. The problem definition phase has three subphases.

3.1.1.1 First Subphase. The first subphase attempts to establish the problem. The problem may be either some existing deficiency, new deficiency brought about by a changing environment, and/or some enhancement to an existing system to increase functionality or productivity. The subphase begins with the user or customer; needs are expressed informally through discussion, or formally through a contract. The process begins with "there's got to be a better way," and leads to "my problem is....," "my constraints are....," and "roughly, what I'm looking for is...." The analysts must identify the problem, including necessary information to be given to--or received from--the system, functions, and performance. The problem statement must clearly distinguish between required functions and optional functionality.

3.1.1.2 Second Subphase. In the second subphase a system analysis is performed to determine the external elements, interfaces, and constraints that may affect the ultimate design. The analysts determine the functions and information flow required to design the solution at a very high level. The analysts may develop prototypes of logical and implementation models to determine the feasibility of finding a solution and better understanding of the problem requirements. These models are defined and described in Section 2.2. The ultimate result of this analysis is a requirements specification document that explicitly states the required functionality. This specification may be a basis from which a contract for system development can be negotiated. This document will also be used by the engineers to formulate solutions.

3.1.1.3 Third Subphase. The last subphase determines if the effort in system development, purchase, training, and maintenance is cost effective. Ultimately, the analysts must decide if the problem is worth the solution. Factors which will impact this decision include a projected cost versus benefits of the solution trade-off, what sort of technology is available, whether development of a new technology is justified, and whether other

alternatives exist (including "doing nothing"). A first cut of division of functions between hardware and software should be made to assess the acceptability of projected performance, reliability, and interfacing.

3.1.2 Software/System Development

The requirements developed in the problem definition phase and documented by the requirements specification documents are implemented in the software/system development phase. In software/system development a solution to the problem is analyzed, designed, implemented, tested, and validated. This phase is divided into the following five subphases that correspond to each of the five activities: requirements analysis, system design, detailed design, test, and Independent Verification and Validation (IV&V). Like the phases of the model, each subphase is typically performed one after another until the entire phase is complete. Exceptions include the prototyping and testing of key portions of the system design to prove viability while reducing risk and the enhancing, optimizing or correcting of previous subphases.

3.1.2.1 Requirements Analysis. The first task for the analysts/engineers is to understand the problem. Two resources are used in doing this: (1) the problem definition and (2) discussions of needs with the problem definition developers. When the customer is the Navy, this discussion can be difficult, since regulations surrounding the contract may restrict the Navy's contact with the contractor. The importance of an unambiguous problem definition is clear in dealing with the Navy, because the concept of the needs and constraints come from that contract. Those needs and constraints may be related to economic, technological, spatial or logistic concerns, or involve training, man-machine interfaces, or integration into existing systems.

Once the analysts/engineers have a clear understanding of the problem, it can be evaluated to determine the communication and processing requirements necessary to obtain the desired results. This evaluation will lead to the specification for the proposed system. Through methods that may involve data-flow diagrams, control-flow diagrams, entity relationship diagrams, and prototyping, the specification identifies functions that the system will perform, but it does not state how the system will be implemented. The analyst must be very careful to avoid including implementation details, such as low-level algorithms and data structure. This hierarchically refined definition of the requirements leads the analysts to logical solutions, the most feasible of which will be determined by user needs and resources. The requirements will be both functional (given input, what should be the output?) and nonfunctional (performance, reliability, etc.). In this stage of development the logical model is used to determine the feasibility of finding a solution or trade-offs between vastly differing solutions. A detailed description of the logical model can be found in Section 4.1.1.1. Prototype implementation models may be developed to show that the process to solve the problem is reasonable.

The final step in this process is the generation of documentation that describes how the various requirements interrelate and delineates possible design solution(s) to meet the requirements. The solution is logical, devoid of all implementation details, and adaptable. It takes into account the fact

that the user's concept of the system changes throughout the development process. The specification also includes validation items used by the designer to show that the final design meets the original requirements.

3.1.2.2 System Design. The next subphase of system development uses the background information, functional requirements, and nonfunctional requirements of the preceding subphase. Depending on the view taken of the data, functions, and needs of the user, one of several methods is followed to determine how the solution should be implemented. This begins with an abstract logical design, which defines the structure of data objects and moves to a concrete implementation model, taking into account network topological and hardware architectural considerations.

3.1.2.3 Detailed Design. During the detailed design subphase, low-level algorithms are selected and implemented. This selection includes algorithms that are implemented in hardware processors and software procedures. Software algorithms are described in a form that emphasizes structured programming constructs. Program Description Language (PDL) may be used to define, in detail, what processing is necessary to implement the algorithm. Programming language details are kept at a minimum so that the PDL can be converted to whatever compilable form is required, although information structure and user requirements may make this impossible. After the system is written in PDL, coding is accomplished with ease, knowing that language detail problems in one module can be addressed separately from other modules.

3.1.2.4 Testing and Integration. During the testing and integration subphase, each module is tested for correctness as a unit. Test cases for the module are designed to ensure proper performance and efficiency. The effects on performance at the limits of design and erroneous input are examined. The planning for such tests begins well before testing actually takes place, as soon as the expected results can be determined.

As the modules of some subfunction of the system are shown to perform as expected in a unit test, they are integrated and a test is applied to the integrated modules. One by one the subfunctions are combined and tested for overall system functionality and interfacing. This can be performed in either a top-down fashion, with code stubs substituting lower level modules, or bottom-up, where drivers are written for groups of elemental modules, or a combination of the two. It is important that careful consideration be given to the test code; the test is valid only if the stubs and drivers are correct. During integration testing, critical modules (those which satisfy many requirements or contain complex code) should be given special attention, testing all paths above and below them.

3.1.2.5 Independent Validation and Verification (IV&V). Validation is the comparison of the integrated system against the requirements as determined during analysis. Verification is the comparison of actual response and results of the system to what is expected. For Navy applications, these are typically performed by an independent contractor. A test plan is drawn up describing how and when the tests will be applied, and a test procedure is written specifying which test will be used to show conformance to which requirement. After IV&V, the Navy users, for whom the system was designed, may do application specific testing (in industry, this is sometimes called a

"beta test"). A system that has been properly analyzed, designed, and tested should pass this process. Since the meeting of requirements involves both ends of the development process, inconsistencies are difficult to correct. Traceability is imperative: if the development process cannot be traced back to the requirements, there is no hope for discovering where the omission occurred or where to begin redesign.

3.1.3 Software System Maintenance (Change Control)

All software, no matter how well designed and coded, requires maintenance. This maintenance can take many forms, not all of which are concerned with correcting errors (corrective). Updates due to changing environments (adaptive) and upgrades to increase functionality or performance (perfective) are two other activities that constitute maintenance and are guaranteed to occur. Since software does not break or wear out, the bulk of maintenance required (once the system has been thoroughly tested and debugged) is adaptive and perfective. It is because of this that the task of system maintenance is often generically called "change control." Maintenance takes the form of prerelease corrections or additions as a result of a review (changes prompted by software trouble reports submitted by the user), or updated and upgraded versions. In this way, the software stays up-to-date with technology and user needs. Changes to the system may involve backing up the development ladder, all the way to the analysis stage. It is imperative that traceability exists throughout the development process so that the impact of changes can be minimized.

3.1.4 Support Activities

The support activities bolster the other three phases of the life cycle and include project management (cost estimation, resource allocation, work scheduling), analysis and design reviews, and document generation. These activities are the backbone of the development process because the other phases require the support to be successful (create a system that works and is maintainable within cost restrictions). One of the most important support activities that impacts all other phases of the development process is configuration management.

3.1.4.1 Configuration Management. Configuration management starts at the beginning of the development process and continues throughout the life of the system. It is based on the fact that change occurs not only after the system has been delivered, but all during the development process. Accepting this fact, it is best to have an orderly and accountable way to manage it. All the following are configuration items: analyses, specifications, design documentation, code listings, test plans, procedures, results, installations, operations, and maintenance manuals. When the development process starts, several versions of a configuration item may exist. When one is selected for use, this becomes part of the baseline configuration. Formal change procedures must be followed to make changes in the baseline. This way, errors or improvements are made and documented in an orderly fashion, and accountability is preserved.

3.2 EXPERT DESIGN ADVISOR METHODOLOGY

The EDA methodology is an automatable process by which a systems engineer can optimize a particular system's design structure. This includes making optimal allocations of logical design objects to physical implementation resources. The EDA methodology assists the engineer in the development of the logical and implementation design as well as in the generation of software code. The methodology is most effective when automated as much as possible. It attempts to provide the systems engineer with a better understanding of the system, and it facilitates the engineer's trade-offs and/or modifications of the system, enhancing his/her ability to make intelligent systems design decisions.

Portions of the EDA methodology can be applied to all stages of system developments. In the problem definition and requirements analysis stages of development, the logical and implementation models may be only throwaway prototype models, used to determine the feasibility of finding a solution, or to trade off between vastly differing solutions. In the design phase, EDA is most useful where it will help the systems engineer find a near optimal design given the system's requirements; e.g., the engineer is able to eliminate poor designs without the cost of prototyping the designs in software and/or hardware. The methodology includes additional assistance to the engineer in the detailed design phase during code development. In the test and IV&V phases, the EDA methodology is useful in optimizing design error corrections found in these phases. Finally, in the change control maintenance phase, the EDA is vital when modifications involve design changes and/or the physical implementation architecture.

The methodology currently requires a data or control-oriented representation of the (possible) logical design and/or a description of the (possible) implementation architecture. Use of object-oriented design descriptions will be added to the methodology at a later date.

The results of the methodology include analysis, refined logical models, refined implementation models, mappings of logical model views to implementation model views, and code generation structure.

EDA methodology is divided into the following sections: System Characterization, Fragmentation and Decomposition, Recomposition, Resource Allocation Mappings, Code Generation, and Automation Opportunity of Methodology.

3.2.1 System Characterization

Characterizations of the system are critical to the methodology. The characterizations are dependent on different attributes of the system. They are developed for the logical model, implementation model, or a combination of both. The characterization of a model may only partially describe the system.

A "view" of a model is defined as a collection of modules (or resources) which completely (necessary and sufficient) describes the system. Figure 3-2 shows two examples of two views that describe a simple system. Note that a view completely spans the decomposition tree such that each path between a leaf node and the root of the tree pass through exactly one node.

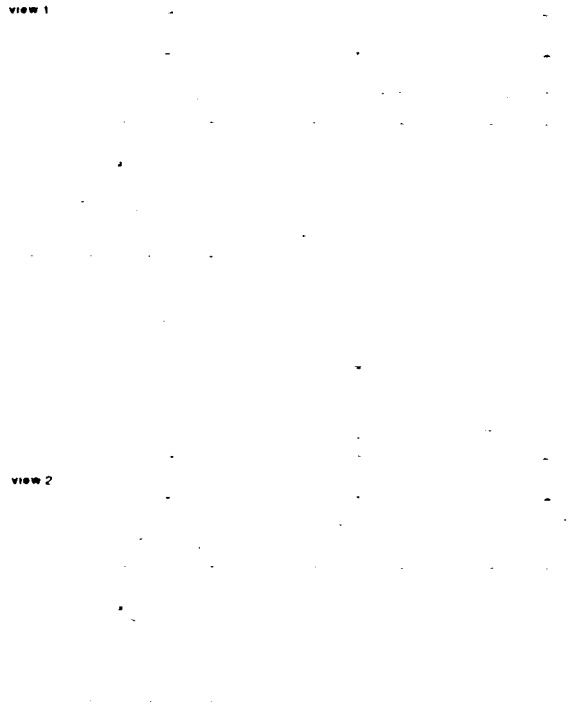


FIGURE 3-2. EXAMPLES OF TWO VIEWS THAT DESCRIBE
A SIMPLE SYSTEM

The major characterizations within EDA are:

- o **IMPLEMENTATION FRAGMENT CHARACTERIZATION**--looks at an incomplete portion of a view of the implementation model.
- o **IMPLEMENTATION LEVEL CHARACTERIZATION**--looks at a complete view of the implementation model.
- o **IMPLEMENTATION MODEL HIERARCHY CHARACTERIZATION**--includes all hierarchical views of the implementation model.
- o **LOGICAL FRAGMENT CHARACTERIZATION**--looks at an incomplete portion of a view of the logical model.
- o **LOGICAL LEVEL CHARACTERIZATION**--looks at a complete view of the logical model.
- o **LOGICAL MODEL HIERARCHY CHARACTERIZATION**--includes all hierarchical views of the logical model.
- o **SYSTEM FRAGMENT CHARACTERIZATION**--looks at only an incomplete portion of a view of the implementation model and an incomplete portion of a view of the logical model.
- o **SYSTEM LEVEL CHARACTERIZATION**--looks at one complete view of the implementation model and one complete view of the logical model at some level.
- o **SYSTEM HIERARCHY CHARACTERIZATION**--includes complete system hierarchy of both logical and implementation models.

3.2.1.1 Attributes. The characterizations are made according to a number of attributes. These include size, complexity, inherent parallelism, the real-time criticalness or support, module/resource dependency, physical support of logical functions, and physical connectiveness. The collection of rules to quantify the attributes is used by the methodology to characterize the logical model, implementation model, and overall system--in conjunction with the MOE environmental parameters, and user model direction.

3.2.1.1.1 **SIZE**. The size of the system has a large impact on the amount of resources necessary to find an optimal allocation. The engineer can determine the relative size of the current problem (view in the system). A size rule for the logical model view is based on the number of logical modules; a size rule for the implementation model view is based on the physical nodes; and a size rule for the overall system view is based on both the logical modules and the physical nodes.

3.2.1.1.2 **COMPLEXITY**. The complexity of the overall system has a large impact on the recomposition, fragmentation/decomposition, and load balancing of the logical and implementation models to satisfy requirements. The complexity rule of a logical system view within the methodology is based on the McCabe complexity analysis equation which, in turn, is based on the number of communication arcs and the number of logical modules. A more complex

measure could be substituted by the engineer for this measure. A complexity rule for the implementation model view is based on the physical node and the connectivity of the implementation nodes. The complexity rule for the overall system is based on the complexity of both the logical model view and the implementation model view.

3.2.1.1.3 PARALLELISM. The parallelism of the system has a large impact on the scheduling, load balancing, and reconfiguration of the logical and implementation model. The parallelism rule for a logical model view is based on the number of modules that can process concurrently with each other. The parallelism rule class for an implementation model view is based on the total parallel threads of the physical system. The parallelism rule class for the overall system is based on the parallelism of both the logical model view and the implementation model view.

3.2.1.1.4 HARD REAL-TIME. The hard real-time nature of a system has an enormous impact on scheduling, load balancing, reconfiguration, and trade-offs among other factors such as reliability, fault tolerance, cost efficiency, etc. The hard real-time rules for the logical model view are based on the criticalness of tasks and the deadline driven characteristic of the logical system. The hard real-time rules of the implementation model view are based on a real-time support characteristic. The hard real-time rules for the overall system are based on the amount of hard real-time support of the implementation model view versus the logical model view needs.

3.2.2 Fragmentation and Decomposition

Fragmentation is the splitting of one module (or one node) into multiple modules (or multiple nodes) within a level of the hierarchy; decomposition is the splitting of one module (or one node) into multiple modules (or multiple nodes) at the next level down in the hierarchy. In the first case, fragmentation, the original module (or node) is replaced, while in the latter, decomposition, the original module (or node) remains in the hierarchy.

The engineer needs to fragment and/or decompose both the logical design models and implementation models as he/she develops a more detailed description of the system. Various attributes in the system determine the amount of decomposition and/or fragmentation. For instance, a typical human can only understand up to seven to nine modules at one time; therefore, an engineer should not decompose (or fragment) a module into more than seven to nine "submodules." Additionally, implementation resources described in the implementation model can be fragmented and/or decomposed into smaller implementation resources based on the physical attributes and characteristics in order to satisfy the requirements and modeling direction. It is important that during fragmentation (the split at one level) the decomposition at higher levels remains consistent. The fragmentation and decomposition should also try to reduce complexity within a level (or a portion of a level) and make the views of the system intuitive.

3.2.3 Recomposition

The purpose of the recomposition is to redefine or merge a portion of the hierarchy, composed of multiple modules and connections, in a new way so that the pertinent information is maintained. It may span levels of the design hierarchy. Recomposition typically tries to make the system more understandable or tries to optimize one model (implementation or logical) to the other model. Figure 3-3 gives an example of a recomposition of a graph. The first graph's A, B, and C modules have been reorganized so that in the second graph the functions in A, B, and C are performed by modules X and Y. Note as in the fragmentation and decomposition, the engineer (with supporting tools) must ensure that consistency is maintained. The recomposition is typically based on the attributes and characteristics of the existing model. Recomposition can occur in both the logical and implementation model.

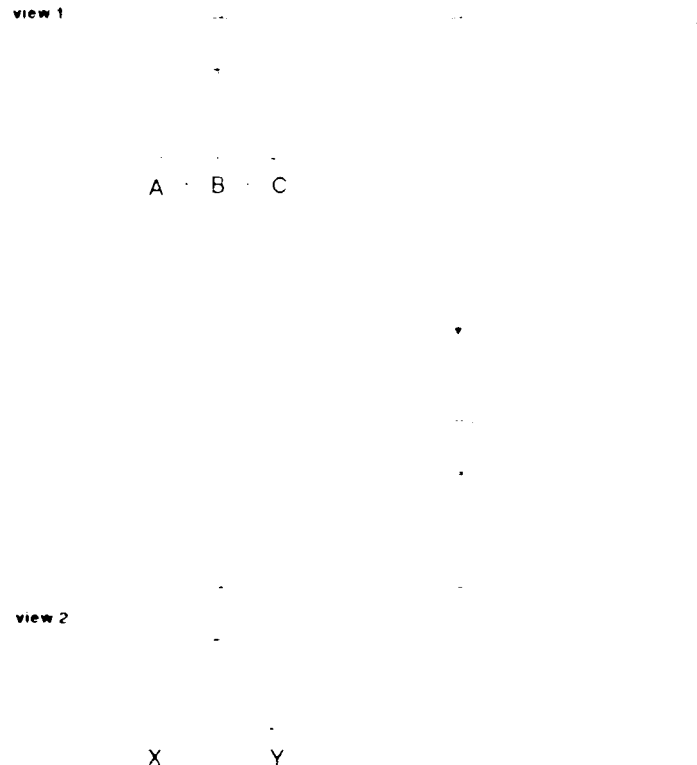


FIGURE 3-3. RECOMPOSITION OF A GRAPH

3.2.4 Resource Allocation Mappings

The purpose of the resource allocation is to allow the mapping (or allocation) of one view of the logical model onto one view of the implementation model, given the environmental parameters so that the system's MOE will be adequately met. The resource allocation methodology typically is applied to one view of the system at one time; though used iteratively, it allows the user to compare multiple system views and/or alternative system definitions.

It is not feasible (even for a rather small system) to search through all possible combinations or permutations of allocations to find the optimal allocation. Therefore, automation support is critical for this portion of the methodology. The methodology attempts to rule out a large sector of bad allocations; then it divides and conquers a small sector of allocations that are in the neighborhood of optimal. Next the resource allocation portion of the methodology specifies the application of a series of mathematical analyses and ratings to determine the most suitable allocation candidates that are in the neighborhood of optimal. This method solves the problem very quickly, more efficiently, and it is better managed for large, complex mission critical systems.

This methodology will not necessarily find the "optimal" solution given the user's MOE. If the search space is too large, the system will produce a "near optimal" solution within the time constraints of the engineer. The engineer can tailor the depth of the methodology according to his/her available resources (computer time, manpower, clock time, etc.).

Figure 1-2 provides an overview for the resource allocation methodology. The methodology uses a number of rules of thumb that are automated into an inference engine and a set of rules. The inference engine is the intelligent mechanism of the system. It is composed of collections of rules (stored in the Rules Library), classes of algorithms (stored in the Algorithms Library), analytical equations, and intelligent probabilistic reasoning, which are the strategies needed to solve the allocation problem.

The resource allocation mapping process includes selection, application, and assessment of allocation algorithms.

3.2.4.1 Allocation Algorithms. A variety of mathematic and heuristic algorithms can be used to generate candidate resource allocation mappings. This section describes some of the kinds of algorithms that might be employed and describes useful ways of categorizing them.

3.2.4.1.1 CLASSES OF ALGORITHMS. The collections of algorithms that are used by the methodology to determine optimal allocations currently consist of fifteen classes: analysis-oriented, communication-oriented, complex analytical equations, computation-oriented, dynamic priority scheduling base, heuristic algorithm, intelligent probabilistic reasoning, mixed analysis- and simulation-oriented, mixed computation- and communication-oriented, nonpreemptive static priority scheduling base, nonrandomized, preemptive static priority scheduling base, randomized, simulation-oriented, and static scheduling base. The selection of both the particular class and algorithm is

based on the attributes of the algorithms; the characterization of the logical, implementation, and system view; system inputs and requirements; and the engineering resource constraints (CPU time, disk space, clock time, etc.).

Provided below is a list of classes of algorithms and their different strategies that optimize resource allocations based on specific criteria.

- o Analysis-Oriented--uses certain analytical equations and rating schemes that are selected for different criteria.
- o Communication-Oriented--minimizes the data volume communication traffic between logical modules. (Refer to Appendix)
- o Complex Analytical Equations--uses equations that were derived from scientific books and research journals.
- o Computation-Oriented--performs load balancing on the system's computational intensity.
- o Dynamic Priority Scheduling Base--schedules tasks based on priorities which are issued during run-time.
- o Heuristic Algorithm--uses information from past experience techniques and probabilistic reasoning.
- o Intelligent Probabilistic Reasoning--uses different probabilistic distribution functions and voting schemes.
- o Mixed Analysis- and Simulation-Oriented--uses a combination of both certain analytical equations and rating schemes, and different simulation techniques at different levels of accuracy that are selected for different criteria.
- o Mixed Computation- and Communication-Oriented--minimizes the data volume communication traffic between logical modules and performs load balancing on the system's computational intensity.
- o Nonpreemptive Static Priority Scheduling Base--schedules tasks based on priorities that are issued prior to run-time with the condition that the tasks cannot be discarded from the execution queue for any reason.
- o Nonrandomized--uses specific patterns of allocation schemes based on the specific objective function(s).
- o Preemptive Static Priority Scheduling Base--schedules tasks based on priorities which are issued prior to run-time and with the condition that the tasks can be discarded from the execution queue if more important tasks need to be executed first.
- o Randomized--uses random allocation schemes based on the specific objective function(s).

- o Simulation-Oriented--uses different simulation techniques and different levels of accuracy that are selected for different criteria.
- o Static Scheduling Base--schedules tasks prior to run-time and based on different types of criteria.

3.2.4.2 Allocation Algorithms Selection. The objective of the Allocation Algorithms Selection is to select the best algorithm from the available library of algorithms to effectively find the best possible solution. This includes solving the problem quickly, accurately, and as near-optimally as possible. The process involves first determining the domain of the problem, the size of the problem, and the criteria that the developer wishes to optimize. This information is used to determine the most suitable algorithm in that domain to solve the problem.

3.2.4.3 Application of Algorithm or Development of Mapping. The application of an algorithm is best described through a sample allocation problem as shown in Figure 1-1. This demonstration is included in order to show the functionality of EDA. The inputs on the left-hand side of the figure include: hardware configurations, properties, attributes, and characteristics, (such as connectivity, communication bandwidth, processing capability, etc.); software logical modules, property attributes and characteristics, (such as complexity, data dependency, parallelism, criticalness, communication intensity, computation intensity, etc.); MOE (such as logical task response time, logical model response time, implementation resources response time, implementation model response time, overall system response time, etc.); environmental parameters (such as reliability and fault tolerance, efficiency, etc.); and user modeling direction (such as assigning specific reliability value, fault tolerant value, critical value, response time to a specific logical task, implementation resource, overall system, etc.).

Based on the characteristics of the given inputs, EDA allocates logical modules as shown in Allocation 1, Figure 1-1.

3.2.4.4 Assess Allocation. The allocation is evaluated based on the simulation reports in comparison with the MOE requirements, environmental parameters and user modeling direction. The report that is generated as a result of simulating the allocation includes information such as performance, reliability, fault tolerance, cost efficiency, etc.

Figure 1-1 shows the iteration of the methodology given that the assessment determines the MOE have not been sufficiently met. Since Allocation 1 does not satisfy the requirements, the engineer (with automation support) reallocates the functions to produce Allocation 2, taking Allocation 1 into consideration. Since Allocation 2 satisfies the requirements, the mission is accomplished. To push the system performance further, the engineer may assign a shorter response time on the overall system, forcing another iteration (Allocation 3), taking Allocations 1 and 2 into account. This process may be repeated until the "optimal" limit is reached.

3.2.5 Code Generation

The purpose of the code generation methodology is to analyze the logical and the physical models' attributes and characteristics and determine the developer's optimal structure of programs needed to be generated in order to satisfy the system's requirements and the user modeling direction.

3.2.6 Automation Opportunity of Methodology

The methodology is automated to a large extent through the use of the intelligent mechanism's knowledge-base which is enhanced as these entities expand. These entities operate on the logical model, implementation model, MOE, environmental parameters, and automation user direction to arrive at the necessary results. The methodology will be automated into five tools. The learning mechanism of EDA is based on the evaluation of the candidate allocation. For example, an inference engine composed of collections of heuristic classes of rules, classes of algorithms, sets of mathematical analysis equations, and sets of intelligent probabilistic reasoning could be used to automate the process. (This automation is described in Chapter 4.)

CHAPTER 4

PROTOTYPE TOOLSET IMPLEMENTATION

The EDA prototype toolset, as implemented, uses a combination of the rule-based artificial intelligence methods and brute force computational intensive algorithmic methods to arrive at its recommendations. The toolset consists of five tools: the Recomposition Advisor (RA) tool, the Fragmentation and Decomposition Advisor (FDA) tool, the Code Generation Advisor (CGA) tool, the Utility tool, and the Resource Allocation Advisor (RAA) tool. EDA is implemented by using Knowledge Engineering Environment (KEE) and Ada in each of the five tools.

The RA analyzes the logical and the implementation models' characteristics and advises the user on recomposing the fragmentation of modules (nodes) into larger modules (nodes) in order to satisfy the requirements. The FDA analyzes the logical and the implementation models' characteristics and advises the user on decomposing the logical modules (nodes) into smaller fragments in order to satisfy the requirements. The CGA analyzes the logical and the implementation models' characteristics and advises the Code Generator and the user on the type of programming structure (i.e., package information) to generate. The Utility tool consists of a collection of routines that are used by the EDA prototype toolset to manipulate and transport the inputs, outputs, and system requirements. The RAA analyzes the logical models' and the implementation models' characteristics and advises the user on the mapping of logical modules onto physical implementation nodes in order to satisfy the requirements.

Architecturally, the RAA tool is the most important to the project. It is currently the only tool implemented and will be used as a prototype for the other tools. The implementations of the RAA emphasize the EDA environment, EDA capability, and EDA usage.

4.1 RESOURCE ALLOCATION ADVISOR TOOL

The RAA performs necessary characterizations which access criteria for determining when other tools are needed to aid the modeling process. This iterative process results in achieving an optimal solution. The RAA functionality provides the user with a set of automated displays and commands in order to manipulate the system representation for an optimal allocation.

4.1.1 System Representation

System Representation consists of hierarchical levels of modules and nodes which represent logical and implementation models. The records and recursive pointers of this hierarchical structure form a powerful logical and implementation model framework. It is the key to robustness and flexibility.

4.1.1.1 Logical Model's Information. The information necessary to represent the logical model includes its properties, attributes, and characteristics captured in modular form. The structure of this representation is designed so that it could be enhanced or expanded as more properties, attributes, and characteristics are defined.

Each module contains the following information: module name, the unique identification of the module; process type, the type of processing that the module performs; connectivity information (data and control), modules that execute prior to or upon completion of any specific module; data that is passed prior to or upon completion of any specific module; operation list, the list that contains instructions used to manipulate data; submodule list and super module list, the lists that contain modules that are part of the logical model hierarchy; parallel process, a list that contains modules that can execute in parallel; real-time, the list that contains timing information such as soft and hard deadlines; and periodicity of the module.

4.1.1.2 Implementation Model's Information. The information necessary to represent the implementation model includes its properties, attributes, and characteristics captured in node form. The structure of this representation is designed so that it could be enhanced or expanded as more properties, attributes, and characteristics are defined.

Each node contains the following information: node name, unique identification of the node; node type, whether the node is a single terminal or a network of terminals; number of parallel threads that contains the number of parallel processes that the node is capable of; lists of subnodes and supernodes that contain other nodes that are part of the implementation model hierarchy; and hardware specs that contain information related to the particular implementation of the node.

Accompanying the node information is the connection information that contains the following information: channel name, a unique communication channel identification; data rate, the communication bandwidth of the channel; data latency, the communication latency of the channel; protocol overhead, the protocol overhead time for this channel; and protocol specs, the protocol specification for this channel.

4.1.2 Functionality

The functionality of the RAA is carried out in three phases: the Rules Selector (RS), the Algorithm Selector (AS), and the Algorithm Applier (AA). These phases are facilitated by the displays and commands as described in Section 4.1.3.

4.1.2.1 Rules Selector. The RS analyzes and evaluates the problem to be solved. Its purpose is to characterize the logical model, implementation model, MOE requirements, environmental parameters, and user modeling direction, each separately and together as a whole system. The results of these characterizations include the selection of classes of rules to be applied to certain algorithms which will be used by the AS.

4.1.2.2 Algorithm Selector. The purpose of the AS is to use the classes of rules given by the RS to reject certain classes of algorithms and select the most suitable algorithm for the AA. It begins the rejection process by rejecting certain algorithms based on the attributes and the characteristics of the logical, physical, and overall system. It starts a second round of matching by rejecting other sets of algorithms using comparison models. By analyzing algorithms left in the library, the selector performs certain ratings based on a class of algorithm types. Finally, it selects the algorithm with the highest rating and recommends it to the user.

4.1.2.3 Algorithm Applier. The purpose of the AA is to take the algorithm given by the AS and run that algorithm to produce an allocation candidate for simulation. The AA executes algorithms by applying rules and generates resource allocation assignments. The AA process begins by taking the recommended algorithm, making a remote procedure call on the VAX to execute that algorithm, and receiving a report of the allocation.

4.1.3 Displays and Commands

EDA is composed of many display panels and commands which ease the resource allocation architectural development process. Each of the tools has different display panels and commands to serve its purpose. Both independently and together, the tools serves EDA's functionality. The RAA displays and commands consist of the following three panels: the System Characterization Panel, the Algorithm Applier Panel, and the Candidate Assignment Display Panel.

4.1.3.1 System Characterization Panel. The System Characterization Panel shown in Figure 4-1 allows the user to evaluate characteristics of the logical, physical, and overall logical and physical systems that are characterized by the EDA's inference engine--and to verify if the characteristics are satisfactory. The user then either accepts the particular characteristics or overrides them if they are not satisfactory.

In the current version of EDA, the System Characterization Panel consists of fourteen attributes that are divided into three different types: logical system, physical system, and overall system. There are seventeen characteristics which are also divided into the same three types; and there are seven commands.

Knowledge Bases
RESOURCE ALLOCATION
System KB's

KEE Desktop 1 - Lisp Listener
Command:

iii (Output) The Graph of the RESOURCE ALLOCATION Knowledge Base

LOGICAL SYSTEM CHARACTERIZATION	PHYSICAL SYSTEM CHARACTERIZATION	OVERALL SYSTEM CHARACTERIZATION
ENTER ALGORITHM APPLICATION SCREEN	SELECT FUNCTION TO PERFORM	RETURN TO TOP LEVEL MENU
ENTER CANDIDATES ASSIGNMENT SCREEN		ENTER HISTORIES SCREEN
ACTIVATE LOGICAL SYSTEM UPDATE		ACTIVATE PHYSICAL SYSTEM UPDATE

LOGICAL SYSTEM ATTRIBUTES	PHYSICAL SYSTEM ATTRIBUTES		OVERALL SYSTEM ATTRIBUTES
NUMBER OF SOFTWARE MODULES 10	NUMBER OF NODES 5	NUMBER OF COMM. LINKS 5	PARALLELISM VALUE 13.36364
NUMBER OF SOFTWARE PROCESSES 10	NETWORK COMPLEXITY VALUE 5	PHYSICAL TOPOLOGY STAR	COMPLEXITY VALUE 15
MCCABE COMPLEXITY 7	CONNECTIVENESS 5	PARALLELISM VALUE 12.0	LOGICAL PLUS PHYSICAL SIZE VALUE 15
PARALLELISM VALUE 1.3636364			OVERALL LOAD DISTRIBUTION VALUE 2.6

LOGICAL SYSTEM CHARACTERIZATIONS	PHYSICAL SYSTEM CHARACTERIZATIONS		OVERALL SYSTEM CHARACTERIZATIONS
HARD REAL-TIME CHARACTERIZATION NOT IMPORTANT	HARD REAL-TIME SUPPORT GOOD	NETWORK COMPLEXITY SIMPLE	SIZE BY LOGICAL PLUS PHYSICAL VERY SMALL
DEPENDENCY CHARACTERIZATION HIGH DEPENDENT	DEPENDENCY CHARACTERISTIC VERY LOW DEPENDENT		COMPLEXITY CHARACTERIZATION SIMPLE
PARALLELISM CHARACTERIZATION VERY LOW PARALLELISM	PARALLELISM CHARACTERISTIC VERY HIGH PARALLELISM		DEPENDENCY CHARACTERIZATION LOW DEPENDENT
MCCABE CHARACTERIZATION SIMPLE	CONNECTIVENESS LOW CONNECTED		PARALLELISM CHARACTERIZATION VERY HIGH PARALLELISM
LOGICAL SYSTEM SIZE SMALL	PHYSICAL SYSTEM SIZE VERY SMALL		PHYSICAL SUPPORT FOR LOGICAL LITTLE IMPORTANT
			OVERALL SYSTEM SIZE VERY SMALL

LOGICAL SYSTEM CHARACTERIZATION PHYSICAL SYSTEM CHARACTERIZATION OVERALL SYSTEM CHARA

10:10 AM 10/14/10 CPU: User Input FREE-STATE's console idle 8 minutes

FIGURE 4-1. SYSTEM CHARACTERIZATION PANEL

4.1.3.1.1 LOGICAL SYSTEM ATTRIBUTES. The following list describes the attributes of the logical system:

- o **MCCABE COMPLEXITY**--the degree of McCabe complexity that the logical system description possesses (real value).
- o **NUMBER OF SOFTWARE MODULES**--the total number of modules within the logical system (integer value).
- o **NUMBER OF SOFTWARE PROCESSES**--the total number of processes that the logical system possesses (integer value).
- o **PARALLELISM VALUE**--the degree of parallelism (average number of parallel threads) that the logical system description possesses (real value).

4.1.3.1.2 PHYSICAL SYSTEM ATTRIBUTES. The following list describes the attributes of the physical system:

- o CONNECTIVENESS--the degree of connectivity that the physical system possesses (the ratio of communication links over the physical size, real value).
- o NETWORK COMPLEXITY VALUE--the degree of complexity that the physical system configuration possesses (certain weight on number of nodes and number of arcs, real value).
- o NUMBER OF COMMUNICATION LINKS--the total number of communication arcs between the nodes that the physical system configuration possesses (real value).
- o NUMBER OF NODES--the total number of physical system nodes (integer value).
- o PARALLELISM VALUE--the total number of parallel threads the physical system possesses (real value).
- o PHYSICAL TOPOLOGY--the type of connection in which the physical system is configured (linear, ring, star, fully connected, hyper-cube, etc.).

4.1.3.1.3 OVERALL SYSTEM ATTRIBUTES. The following list describes the attributes of the overall system:

- o COMPLEXITY VALUE--the overall degree of complexity of the logical and physical system (real value).
- o LOGICAL PLUS PHYSICAL SIZE--the overall size in terms of the total number of logical modules and physical nodes (integer value).
- o OVERALL LOAD DISTRIBUTION--the overall distribution of the logical modules to physical nodes (integer value).
- o PARALLELISM VALUE--the overall degree of parallelism of the logical and the physical system (real value).

4.1.3.1.4 LOGICAL SYSTEM CHARACTERIZATION--The following list describes the characteristics of the logical system:

- o DEPENDENCY CHARACTERIZATION--the logical system's data dependency characteristic is either independent, very low dependent, moderate dependent, high dependent, very high dependent, extremely dependent).
- o HARD REAL-TIME CHARACTERIZATION--the logical system's real-time characteristic (modules' dead line, processing, etc.) is either not important, little important, important, very important, extremely important, critically important).

- o **LOGICAL SYSTEM SIZE**--the logical system's size characteristic is either tiny, very small, small, medium, large, very large, huge).
- o **MCCABE CHARACTERIZATION**--the logical system's McCabe complexity characteristic is either very simple, simple, moderate, complicated, very complicated, extreme.
- o **PARALLELISM CHARACTERIZATION**--the logical system's parallelism characteristic (the rate of modules processing concurrently) is either serial, extremely low parallelism, very low parallelism, low parallelism, moderate, high parallelism, very high parallelism, extremely high parallelism).

4.1.3.1.5 **PHYSICAL SYSTEM CHARACTERIZATION.** The following list describes the characteristics of the physical system:

- o **CONNECTIVENESS**--the physical system's connectivity characteristic system is either extremely low connected, very low connected, low connected, moderately connected, highly connected, very high connected, extremely high connected, fully connected.
- o **DEPENDENCY CHARACTERIZATION**--not implemented yet (may be removed).
- o **HARD REAL-TIME SUPPORT**--the physical system's real-time support characteristic is either very poor, poor, fair, good, very good, excellent.
- o **NETWORK COMPLEXITY**--the physical system's network complexity characteristic is either very simple, simple, moderate, complicated, very complicated, extreme.
- o **PARALLELISM CHARACTERIZATION**--the physical system's parallelism characteristic is either serial, extremely low parallelism, low parallelism, moderate parallelism, high parallelism, very high parallelism, extremely high parallelism.
- o **PHYSICAL SYSTEM SIZE**--the physical system's size characteristic is either tiny, very small, small, medium, large, very large, huge.

4.1.3.1.6 **OVERALL SYSTEM CHARACTERIZATION.** The following list describes the characteristics of the overall system:

- o **COMPLEXITY CHARACTERIZATION**--the overall system's complexity characteristic is either very simple, simple, moderate, complicated, very complicated, extreme.
- o **DEPENDENCY CHARACTERIZATION**--not implemented yet.
- o **OVERALL SYSTEM SIZE**--the overall system's size characteristic is either tiny, very small, small, medium, large, very large, huge.

- o **PARALLELISM CHARACTERIZATION**--the overall system's parallelism characteristic is either serial, extremely low parallelism, low parallelism, moderate parallelism, high parallelism, very high parallelism, extremely high parallelism.
- o **PHYSICAL SUPPORT FOR LOGICAL**--the overall system's physical support for the logical characteristic is either nonfatal, little important, important, extremely important, fatal.
- o **SIZE BY LOGICAL PLUS PHYSICAL**--the overall system's size characteristic is either tiny, very small, small, medium, large, very large, huge.

4.1.3.1.7 **SYSTEM CHARACTERIZATION COMMANDS.** The following list describes the commands of the System Characterization Panel:

- o **ACTIVATE LOGICAL SYSTEM UPDATE**--this command will activate the logical system's set of rules to characterize the attributes and the characteristics of the logical description.
- o **ACTIVATE OVERALL SYSTEM UPDATE**--this command will activate the overall system's set of rules to characterize the attributes and the characteristics of both the physical system and the logical system as an overall system.
- o **ACTIVATE PHYSICAL SYSTEM UPDATE**--this command will activate the physical system's set of rules to characterize the attributes and the characteristics of the physical system.
- o **ENTER ALGORITHM APPLICATION SCREEN**--this command will bring up the ALGORITHM APPLICATION panel.
- o **ENTER ASSIGNMENT DISPLAY PANEL**--this command will bring up the CANDIDATES ASSIGNMENT DISPLAY panel.
- o **ENTER HISTORY PANEL**--this command will bring up the HISTORY panel.
- o **RETURN TO TOP LEVEL MENU**--this command will bring up EDA's top level menu.

4.1.3.2 Algorithm Applier Panel. The Algorithm Applier Panel shown in Figure 4-2 allows the user to evaluate the algorithm recommended by the RAA's Algorithm Selector. It also allows the user to verify if the recommendation is satisfactory by accepting or overriding the selection. If it is rejected, the user has three options: to reconsider different algorithms that have previously been rejected, to manually reject certain algorithms that are not suitable, and to choose other algorithms (which may be suitable) for execution. In this version there are six displays and ten commands.

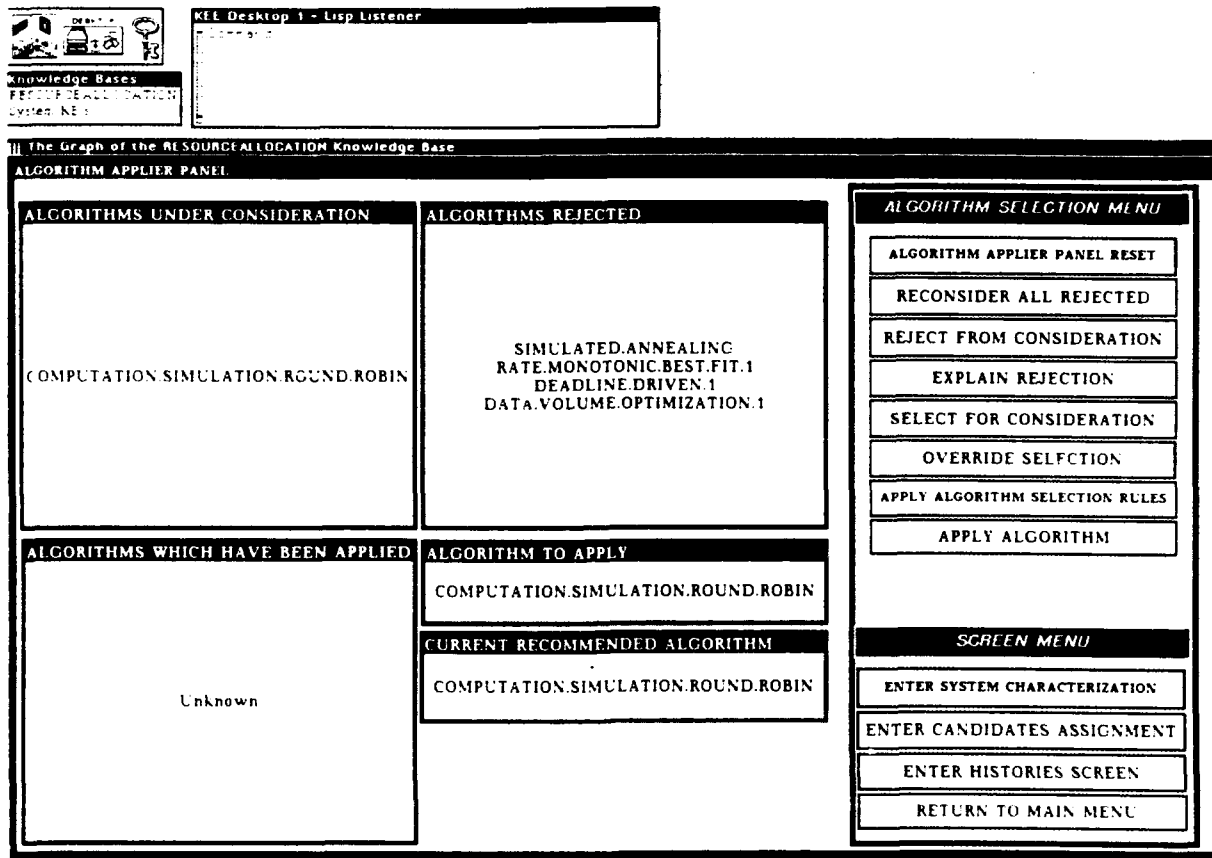


FIGURE 4-2. ALGORITHM APPLIER PANEL

4.1.3.2.1 ALGORITHM APPLIER PANEL DISPLAYS. The following list describes the Algorithm Applier Panel Display:

- o **ALGORITHM TO APPLY**--this display shows the algorithm that has been chosen to be applied.
- o **ALGORITHMS REJECTED**--this display shows the algorithms that have been rejected by the Algorithm Selector from the ALGORITHMS UNDER CONSIDERATION.
- o **ALGORITHMS UNDER CONSIDERATION**--this display shows all algorithms contained in the Algorithm Library that are under consideration.

- o ALGORITHMS WHICH HAVE BEEN APPLIED--this display shows the history of the algorithms that have been applied.
- o CURRENT RECOMMENDED ALGORITHM--this display shows the algorithm that has been recommended by the Resource Allocation Advisor to be applied.

4.1.3.2.2 ALGORITHM APPLIER PANEL COMMANDS. The following list describes the commands for the Algorithm Applier Panel:

- o ALGORITHM APPLIER PANEL RESET--this command will reset all the displays in the panel to the initial states.
- o APPLY ALGORITHM--this command will make a remote procedure call to the VAX for executing the algorithm shown in the ALGORITHM TO APPLY display and receive a report of the candidate allocation.
- o APPLY ALGORITHM SELECTION RULES--this command will activate the classes of rules selected by the RULES SELECTOR to reject unsuitable classes of algorithms and individual algorithms. It then analyzes the algorithms that have not been rejected using mathematical equations, probabilistic reasoning, and heuristic ratings, following this with a set of rules that determines the algorithm which has the highest rating. The algorithm with the highest rating is the recommended algorithm and will be shown in the CURRENT RECOMMENDED ALGORITHM display.
- o ENTER ASSIGNMENT DISPLAY--this command will bring up the CANDIDATE ASSIGNMENT DISPLAY panel.
- o ENTER HISTORY PANEL--this command will bring up the HISTORY panel.
- o ENTER SYSTEM CHARACTERIZATION--this command will bring up the SYSTEM CHARACTERIZATION panel.
- o EXPLAIN REJECTION--this command will list the algorithms contained in the ALGORITHM REJECTED display for the user to select. This selection will generate the explanation for the chosen rejected algorithm.
- o OVERRIDE SELECTION--this command will list the algorithms contained in the ALGORITHMS UNDER CONSIDERATION display so that the user can override the algorithm shown in the CURRENT RECOMMENDED ALGORITHM display.
- o RECONSIDER ALL REJECTED--this command will take all the algorithms that have been rejected in the ALGORITHMS REJECTED display and put them back into the ALGORITHMS UNDER CONSIDERATION display for reconsideration.

- o **REJECT FROM CONSIDERATION**--this command will list the algorithms contained in the ALGORITHMS UNDER CONSIDERATION display for the user to reject. The rejected algorithms will be displayed in the ALGORITHMS REJECTED display.
- o **RETURN TO MAIN MENU**--this command will bring up EDA's top level menu.
- o **SELECT FOR CONSIDERATION**--this command will select the algorithm in the ALGORITHMS REJECTED display and reconsider it.

4.1.3.3 Candidate Assignment Panel. The Candidate Assignment Panel shown in Figure 4-3 allows the user to do the following: view the displays of the distribution and utilization of the allocation in bar chart form; display the allocation on the terminal screen in text form; display the specific physical or logical system individually; and print the allocation. In this version there are five displays and twelve commands.

4.1.3.3.1 **CANDIDATE ASSIGNMENT PANEL DISPLAYS**. The following list describes the Candidate Assignment Panel Displays:

- o **ASSIGNMENT TYPE**--this display shows either the logical type or the physical type.
- o **DISTRIBUTION OF ASSIGNMENT**--this display will show the distribution of the logical modules onto physical nodes in bar graph form.
- o **PHYSICAL UTILIZATION**--this display will show the utilization of the physical nodes in bar graph form.
- o **SINGLE CANDIDATE ASSIGNMENT**--this display will show the allocation mapping for the candidate appearing in the SINGLE CANDIDATE NAME display.
- o **SINGLE CANDIDATE NAME**--this display will show either the name of the logical module when LOGICAL appears on the ASSIGNMENT TYPE display or the name of the physical node when PHYSICAL appears on the ASSIGNMENT TYPE display.

4.1.3.3.2 **CANDIDATE ASSIGNMENT PANEL COMMANDS**. The following list describes the commands for the Candidate Assignment Display Panel:

- o **CANDIDATE ASSIGNMENT RESET**--this command will reset the CANDIDATE ASSIGNMENT panel to its initial state.
- o **DISPLAY ALL AT ONCE**--this command will dump the allocation mapping summary of both the physical system and the logical system onto the output screen.
- o **DISPLAY ALLOCATION FOR A LOGICAL**--this command will show the allocation for the logical module whose name appears in the SINGLE CANDIDATE NAME display via the SINGLE CANDIDATE ASSIGNMENT display.

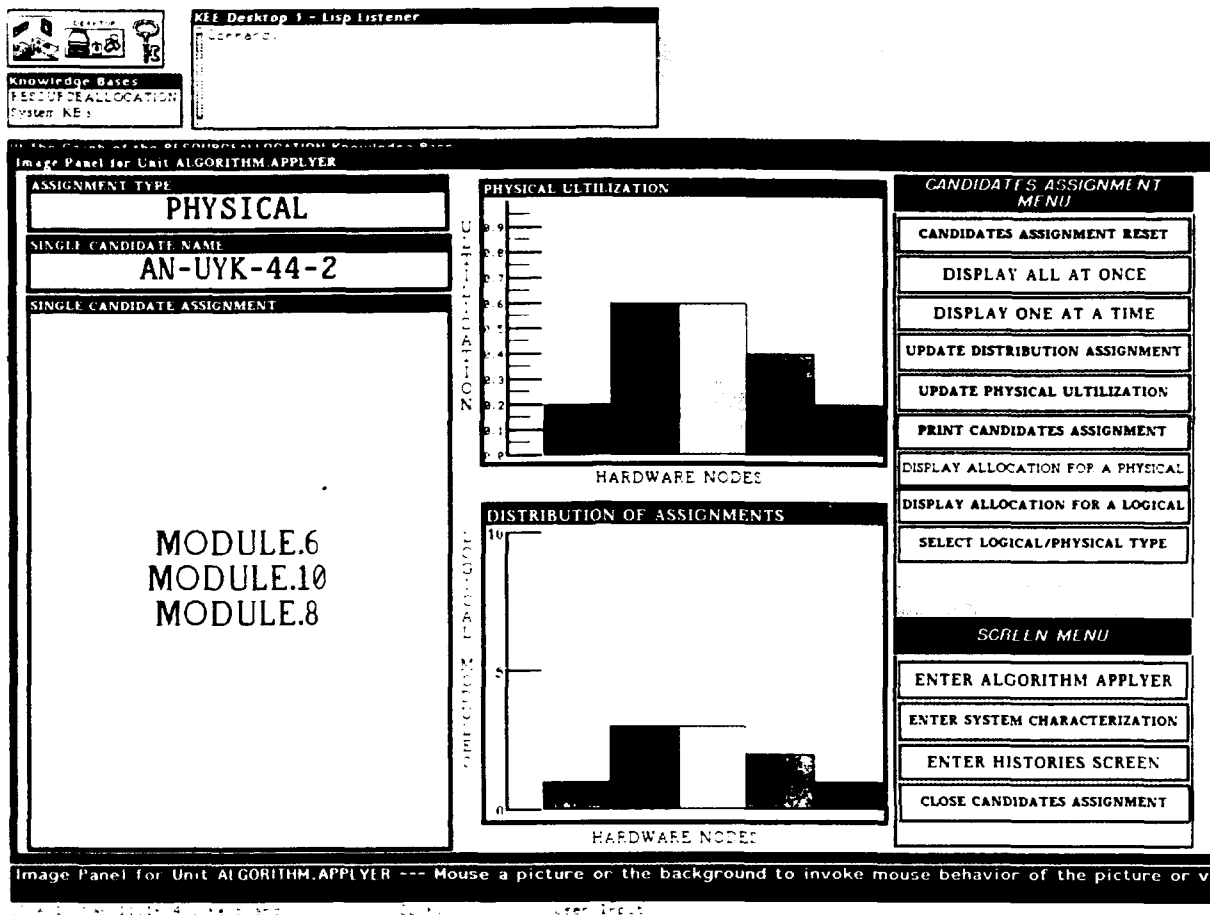


FIGURE 4-3. CANDIDATE ASSIGNMENT
DISPLAY PANEL

- o DISPLAY ALLOCATION FOR A PHYSICAL--this command will show the allocation for the physical node whose name appears in the SINGLE CANDIDATE NAME display via the SINGLE CANDIDATE ASSIGNMENT display.
- o DISPLAY ONE AT A TIME--this command will show the allocation for each logical module or each physical node appearing in the SINGLE CANDIDATE NAME display, one at a time, via the SINGLE CANDIDATE ASSIGNMENT display.
- o PRINT CANDIDATE ASSIGNMENT--this command will print out a summary of both the allocation of the logical system and the physical system.

- o **SELECT LOGICAL/PHYSICAL TYPE**--this command will toggle between LOGICAL and PHYSICAL in the ASSIGNMENT TYPE display.
- o **UPDATE DISTRIBUTION ASSIGNMENT**--this command will show the distribution of the logical system onto the physical system via the DISTRIBUTION OF ASSIGNMENT display in bar graph form.
- o **UPDATE PHYSICAL UTILIZATION**--this command will show the utilization of the physical system via the PHYSICAL UTILIZATION in bar graph form.

4.2 EDA ENVIRONMENT

The EDA Environment consists of networks of computers and the architecture that surrounds the networks. The EDA network, as currently implemented (Figure 4-4), is distributed over two different types of computers, Symbolics and VAX, communicating via ethernet. Each of the computers contains different components which are parts of EDA.

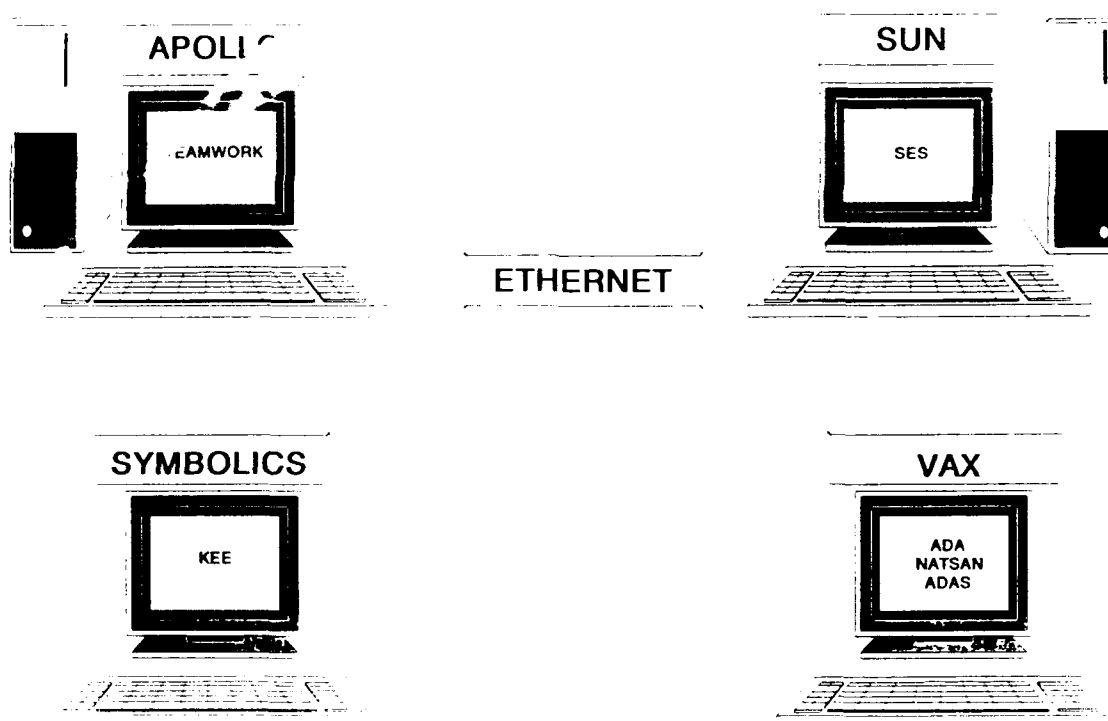


FIGURE 4-4. EDA NETWORK

4.2.1 Symbolics and Knowledge Engineering Environment

The hardware and software environments used to prototype EDA include the Symbolics workstation and the Knowledge Engineering Environment (KEE), a commercial expert system shell. In this version of EDA, the majority of the system software resides on the Symbolics except for a part of the Algorithm Library. The decision-aided mechanism for EDA was based on KEE which has an object-oriented programming capability to capture EDA structure. Because of its inherent properties, KEE provides a flexible environment for a throwaway prototype.

4.2.2 VAX and Ada

The software residing on the VAX computer is part of the Algorithm Library which is used by the RAA to determine the best possible candidate allocations. The algorithms were implemented in Ada on the VAX.

4.3 EDA CAPABILITY

EDA is capable of allocating tasks to resources in order to satisfy the system's requirements. For example, it takes tasks, such as software modules, jobs, and objects, and allocates them to resources, such as networks of computers, humans, and assembly robots, respectively. The toolset is useful for systems engineers, systems analysts, human resource managers, etc. The size and performance play a major role in allocation.

4.3.1 Allocations of Software Modules onto Hardware Resources

EDA can be used to analyze an existing configuration of a computer intensive system. For example: Is the existing allocation of tasks for a certain computer configuration "optimal"? It attempts to determine better allocations for the system as a whole, given the user's requirements for performance, reliability, fault tolerance, and cost-efficiency.

EDA can also be used to analyze a new developing software intensive system and determine an optimal configuration based on the requirements of the system (such as hard and soft real-time reliability factors) before the configuration is implemented. Knowing that the configuration must meet the requirements of the system in advance reduces the cost and time of developing such a system.

4.3.2 Size and Performance

EDA was designed with a highly robust framework and structure. It is capable of supporting an extremely large implementation model as well as an extremely large logical model. However, since the optimal solution to the resource allocation problem is NP-hard (the solution to this type of problem is highly computational complex, e.g., as the size of the problem increases,

the computational complexity increases), the time it would take to determine the optimal allocation would vary according to the size of the system and would increase rapidly (exponentially).

4.4 EDA USAGE

In order to solve the resource allocation problem, the user begins the RAA process which includes inputs, characterization, and allocation. The RAA then proceeds to evaluation and assessment until the system requirement is satisfied. In this prototype version, EDA presents the user with three main panels: System Characterization Panel, Algorithm Applier Panel, and Candidate Assignment Display Panel. The normal usage sequence is to start from the System Characterization Panel, proceed to the Algorithm Applier Panel, and then to the Candidate Assignment Display Panel.

4.4.1 RAA Process

The RAA process begins with the user inputting all the available information and directions. The RAA then characterizes the inputs and determines an allocation which will be evaluated by both the user and simulation tools, iteratively, until the system's requirements are satisfied for assessment.

4.4.1.1 Inputs, Characterization, Allocation. The process shown in Figure 4-5 begins with the user inputting all the requirements and information available such as logical description, implementation resources, MOE and environmental parameters. The RAA then characterizes them both individually and as a whole and presents the characteristics to the user for verification or modeling direction. The user may accept the characteristics analyzed by the RAA or the user may choose to override certain attributes or certain characteristics of the system. Based on the results of this process, the RAA will select certain classes of rules that are most suitable to the problem which will be used to reject both particular classes of algorithms and individual algorithms. Finally, the RAA determines the most suitable algorithm and presents it to the user for verification or modeling direction. The user may accept the presented algorithm evaluated by the RAA, or the user may choose to override it with another algorithm. Once an algorithm is verified, EDA will apply it and present the candidate allocation to the user for evaluation.

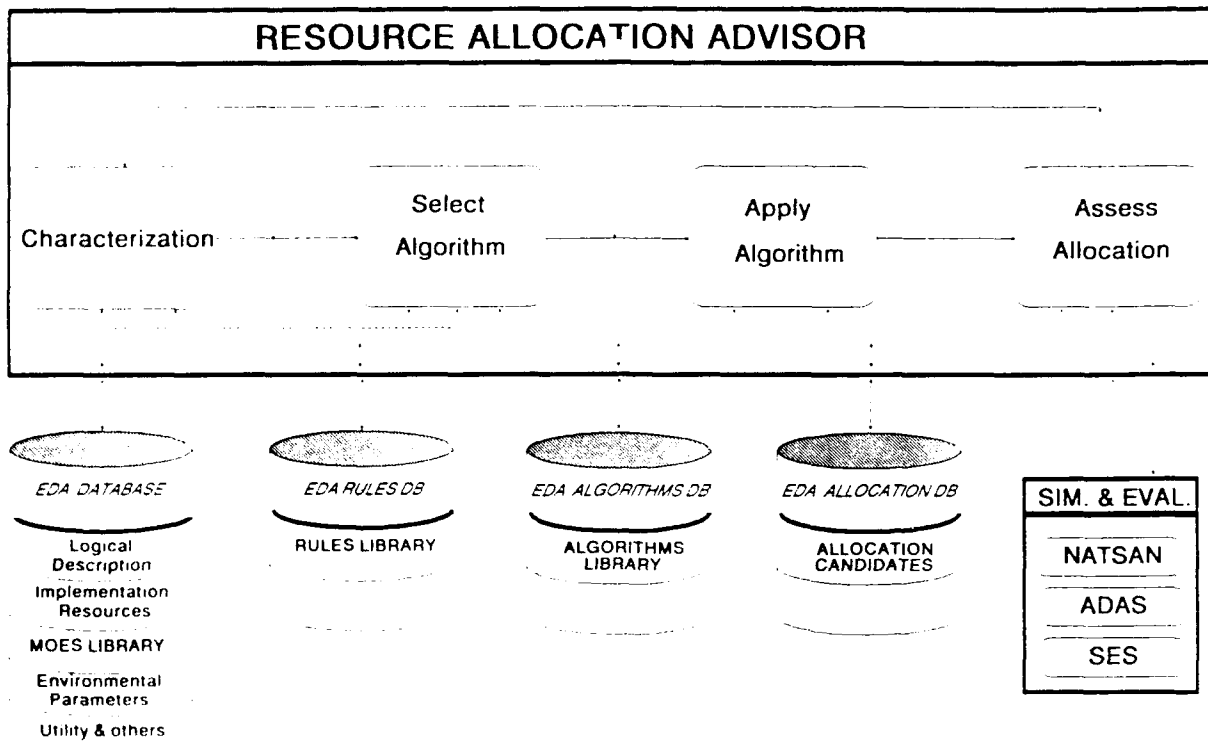


FIGURE 4-5. RAA TOOL FUNCTIONALITY OVERVIEW

4.4.1.2 Evaluations and Assessment. Evaluation is done by simulating the candidate allocation using Scientific and Engineering Software (SES/Workbench), Architecture Design and Assessment System (ADAS), and Navy Tactical Software Analyzer (NATSAN) simulation tools. These tools will simulate the candidate assignment and produce the results for the user and the RAA. The RAA then determines if the results of the allocation satisfies the requirements. If the requirements are satisfied, allocation is assessed, thereby completing the allocation of tasks to resources. However, if the requirements are not satisfied, the RAA will reconsider other algorithms as a feedback lesson, taking the failed allocation, failed classes of rules, and failed classes of algorithms into account. This process is repeated until the requirements are satisfied.

CHAPTER 5

FUTURE PLANS AND DEVELOPMENT

The future plans for the EDA involve enhancing the methodology by expanding and developing the following areas: structures, robustness, major impact parameters, inference engine, RA tool, FDA tool, CGA tool, and Utility tool. In addition, EDA architecture will be integrated with simulation and Case tools to enhance its integrity and portability.

The EDA network will incorporate the Sun and Apollo computers to enhance the portability and capability of EDA on different platforms.

5.1 METHODOLOGY ENHANCEMENTS AND DEVELOPMENTS

The methodology enhancements and developments will include (1) defining new major impact parameters and (2) expanding and developing the RA, FDA, CGA, and support methods and techniques.

5.2 FUTURE VERSION OF TOOLSET

From the lessons learned and experience gained during the development of this version of the toolset, the future version will be developed on a more standardized platform such as Sun. The decision-making mechanism part will be developed using an expert system shell that is less dependent on the platform. The enhancements will include interfaces to more standardized graphics systems and relational data bases. This will enhance the capability of EDA and reduce the transition cost effort.

The next generation prototype will be integrated with other simulation and CASE tools such as NATSAN, ADAS, SES, and Teamwork. NATSAN, ADAS, and SES will be directed by EDA to perform simulation (at different levels) of certain candidate allocations and transmit the results to the user and EDA for evaluation.

5.2.1 Continuation of Work

This effort will continue as part of the Engineering of Complex Systems Technology Block Program. This program will address the design automation problem in the context of large software intensive systems engineering. It will enhance and validate the methods and tools developed under the IED program. A user manual will be developed to facilitate utilization.

BIBLIOGRAPHY

Bianchini, Jr., R. and Shen, J.P., "Interprocessor Traffic Scheduling Algorithm for Multiple-Processor Networks," IEEE Transactions on Computers, Vol. C-36, No. 4, Apr 1987.

Cvetanovic, Z., "The Effects of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Systems," IEEE Transactions on Computers, Vol. C-36, No. 4, Apr 1987.

Ibaraki, T., and Katoh, N., Resource Allocation Problems: Algorithmic Approaches, MIT Press, Cambridge, MA, 1988.

Jamieson, L.H., Gannon, D., and Douglass, R.J., The Characteristics of Parallel Algorithms, MIT Press, Cambridge, MA, 1987.

Lee, S., and Aggarwal, J.K., "A Mapping Strategy for Parallel Processing," IEEE Transactions on Computers, Vol. C-36, No. 4, Apr 1987.

Martin, J., and Oxman, S., Building Expert Systems: A Tutorial, Prentice Hall, Englewood Cliffs, NJ, 1988.

Pearl, J., Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley Publishing Company, Inc., Reading, MA, 1984.

Pearl, J., Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

Peng, D., and Shin, K., "Modeling of Concurrent Task Execution In a Distributed System for Real-Time Control," IEEE Transactions on Computers, Vol. C-36, No. 4., Apr 1987.

NOMENCLATURE

This nomenclature defines terms used in this report. A number of the definitions incorporate acronyms which are defined as part of these terms. Self-explanatory terms carry only the acronym.

Ada--a high level programming language mandated by DoD for embedded systems that stresses modularity, reliability, and maintainability.

Algorithm Applier (AA)--takes the algorithm given by the Algorithm Selector and runs that algorithm to produce an allocation candidate for simulation.

Algorithm Applier Panel--allows the user to evaluate the algorithm recommended by the Resource Allocation Advisor's Algorithm Selector.

Algorithm Selector (AS)--uses the classes of rules given by the Rules Selector to reject certain classes of algorithms and select the most suitable algorithm for the Algorithm Applier.

Analysis-Oriented--the class of algorithms that optimizes the resource allocations by using certain analytical equations and rating schemes that are selected for different criteria.

Anti-Submarine Warfare (ASW)

Candidate Allocation Evaluation--the evaluation process that determines if the candidate allocation satisfies all the system's requirements and Measures of Effectiveness.

Candidate Assignment Display Panel--allows the user to do the following: view the displays of the distribution and utilization of the allocation in bar chart form, to display the allocation on the terminal screen in text form, to display the specific physical or logical system individually, and to print the allocation.

Computer Aided Software Engineering (CASE)--a software development tool used by systems engineers to design, analyze, test, and implement systems.

Code Generation Advisor (CGA) Tool--analyzes the logical modules and implementation nodes and advises the Code Generator and the user on the structure of programs to generate.

NOMENCLATURE (Cont.)

Communication-Oriented--the class of algorithms that optimizes the resource allocations by minimizing data volume communication traffic between logical modules.

Computation-Oriented--the class of algorithms that optimizes the resource allocations by load balancing on the system's computational intensity.

Dynamic Priority Scheduling Base--the class of algorithms that optimizes the resource allocations by scheduling tasks based on priorities which may change during run-time.

Expert Design Advisor (EDA)--a decision aided toolset for use by systems engineers in facilitating the development of large, complex systems involving mission critical computing resources.

Environmental Parameters--the parameters that affect the behavior of the system.

Fragmentation Decomposition Advisor (FDA) Tool--decomposes larger logical modules into many smaller fragments based on the logical attributes and characteristics.

Heuristic Algorithm--the class of algorithms that optimizes the resource allocations by using information from past experience techniques and probabilistic reasoning.

Hybrids--circuits fabricated by interconnecting smaller circuits of different technologies mounted on a single substrate.

Independent Exploratory Development (IED)

Implementation Model Size--the magnitude of the resources topology.

Implementation Model Framework (or physical model framework)--designed in a hierarchical structure to encapsulate all the physical properties, attributes, and characteristics of the implementation model.

Inference Engine--the intelligent mechanism of the system.

Intelligent Probabilistic Reasoning--comprised of a collection of reasons and rules collected and/or derived from textbooks, journals, tools, and researchers. Uses different probabilistic distribution functions and voting schemes.

Logical Model--designed in a hierarchical structure to encapsulate all the logical properties, attributes, and characteristics of the system under design.

Logical Model Complexity--the complexity of the logical model described by the McCabe complexity equation.

NOMENCLATURE (Cont.)

Logical Model Hard Real-Time--the hard dead-line of the logical system.

Logical Model Parallelism--the parallelism of the logical modules within the logical model.

Logical Model Size--the magnitude of the logical model topology.

Logical System Attributes--the attributes that describe different aspects of the logical system.

Logical System Characteristics--enumerated parameters that describe different characteristics of the logical system.

Measures of Effectiveness (MOE)--are typically derived from the requirements and other programmatic sources.

Mission Critical Computing Resources (MCCR)--the computing resources that insure the survivability of the mission.

Mixed Analysis- and Simulation-Oriented--the class of algorithms that optimizes the resource allocations by combining both analytical and simulation orientations.

Mixed Computation- and Communication-Oriented--the class of algorithms that optimizes the resource allocation by minimizing the data volume communication traffic between logical modules and performs load balancing on the system's computation intensity.

Naval Tactical Software Analyzer (NATSAN)--the software tool developed to analyze the complexities of software systems.

Nonrandomized--the class of algorithms that optimizes the resource allocations by using specific patterns of allocation schemes based on the specific objective function(s).

Nonpreemptive Static Scheduling Base--the class of algorithms that optimizes the resource allocations by scheduling tasks that are issued prior to run-time with the condition that they cannot be discarded from the execution queue for any reason.

Overall Model Complexity--the term that describes the complexity of both the logical and implementation models.

Overall System Attributes--the attributes that describe different aspects of both the logical model and implementation model.

Overall Model Parallelism--the dynamic of overall system parallelism.

Overall Model Size--the combination of both the logical and implementation models' size

NOMENCLATURE (Cont.)

Physical Model Complexity--the connectivity of the physical components.

Physical Model Parallelism--the degree of physical components that can process in parallel.

Physical System Attributes--the value of different properties of physical systems.

Physical Model Hard Real-Time--the degree of hard real-time deadline.

Preemptive Static Priority Scheduling Base--the class of algorithms that optimizes the resource allocations by scheduling tasks based on priorities which are issued prior to run-time and with the condition that they can be discarded from the execution queue if more important tasks need to be executed first.

Program Description Language (PDL)--a structured pseudo code language used to describe a program.

Randomized--the class of algorithms that optimizes the resource allocations by using random allocation schemes based on the specific objective function(s).

Recomposition Advisor (RA) Tool--analyzes the logical (or implementation) model's characteristics and advises the user on recomposing the fragmentation of modules (or nodes) into larger modules (or nodes) in order to satisfy the requirements.

Resource Allocation Advisor (RAA) Tool--characterizes the logical and physical models and uses other tools (i.e., Recomposition Advisor, Fragmentation and Decomposition Advisor, and Code Generation Advisor) to recommend the best possible allocation candidate based on the system's attributes and characteristics in order to satisfy the system's requirements.

Rules Selector (RS)--analyzes and evaluates the problem to be solved by characterizing the logical model, implementation model, measures of effectiveness requirements, environmental parameters, and user modeling direction, each separately and together as a whole.

SES/Workbench Tool--a tool used to simulate computer hardware and software system's behavior.

Simulation Oriented--the class of algorithms that optimizes the resource allocations by using different simulation techniques and different levels of accuracy that are selected for different criteria.

Spiral Model--depicts the use of preliminary risk analysis and prototypes along with the standard development process.

NOMENCLATURE (Cont.)

Static Scheduling Base--the class of algorithms that optimizes the resource allocations by scheduling tasks prior to run-time and based on different types of criteria.

Symbolics--a computer specializing in Artificial Intelligence.

System Characterization Panel--allows the user to evaluate characteristics of the logical, physical, and overall logical and physical systems that are characterized by the EDA's inference engine--and to verify if the characteristics are satisfactory.

Teamwork--a CASE tool that automates standard structured methodologies using interactive computer graphics and multi-user workstation power.

User Modeling Direction--comprises the specific tailoring directions from the user in order to assist the intelligent system of EDA.

Utility Tool--a collection of routines that is used by the EDA prototype toolset to manipulate and transport the inputs, outputs, and system requirements.

VAX--a computer system developed by Digital Equipment Corporation.

VHSIC (Very High Speed Integrated Circuit)--a DoD program to develop new generations of silicon integrated circuits that will provide higher performance.

VLSI (Very Large Scale Integrated Circuit)

Waterfall Model--expresses a linear development process beginning with the establishment of requirements and concluding with maintenance.

APPENDIX
SAMPLE ALGORITHM

SAMPLE ALGORITHM

A sample algorithm is provided for illustrating the optimization of communication criteria.

DATA OPTIMIZATION ALGORITHM

The data optimization algorithm will optimize the resource allocations by minimizing the data volume communication traffic between logical modules. By optimizing the data communication alone, the performance of the system is guaranteed to increase.

The algorithm consists of two main steps: characterization and assignment mapping. Taken together, they determine the communication intensity and reduce the communication traffic.

The data optimization algorithm is summarized in the algorithm outline.

Characterization

The characterization step determines the communication characteristics (such as data volume) of the logical system and (such as data rates and data latencies) of the physical system. In this algorithm, both the communication characteristics of the logical modules and hardware nodes are classified as data links. Each link contains two parts: the sending node or module and the receiving node or module. The hardware links are ranked from fastest to slowest. The software links are classified by data volume. Communication channels that transfer more data are given a higher mapping priority.

Assignment Mapping

The assignment mapping step consists of strategies to minimize communication traffic. It relies heavily on how the system is characterized. Assignment mapping looks at how the system is classified by the characterizations and assigns the software modules to the hardware nodes.

The assignment mapping attempts to assign the modules comprising the largest data links to the nodes comprising the fastest data links. If possible, it assigns the modules of a communication link to the same node. This is favorable because the data transfer rate within a node is much faster than the data transfer rate between two nodes. However, it is not always the best method.

If every module were placed on the same processor, none of them could run in parallel, and the target system would not work at its maximum efficiency. Thus, it is necessary to limit the number of nodes that can be

put on each processor. In this case, an even distribution is desired, so the limit of modules on each node was restricted to the number of nodes divided by number of modules, rounding up if the answer is not an integer.

Algorithm Outline

The algorithm outline summarizes the characterization step and the assignment mapping step. This algorithm is coded according to the algorithm outline below:

- I. Create an ordered linked list of software links by data volume (software characterization).
- II. Create an ordered linked list of hardware links by data transfer rate. (hardware characterization)
- III. Create an unordered list of hardware nodes, with a field for assigned software modules. (blank assignment list)
- IV. Create an unordered list of software modules, with a field for assigned hardware nodes. (blank assignment list)
- V. Determine the maximum number of modules to be on each node.
- VI. Begin mapping
 - A. Look at the largest software link
 - B. Are the two modules already mapped?
 1. If yes, do nothing
 2. If both are unassigned:
 - a. Find a node with room on it for two modules
 - b. Assign the modules to the node
 3. If no node has room:
 - a. Find the fastest link with room on either side for a module
 - b. Assign the modules to the appropriate places on each link
 4. If there are no open links, do nothing
 - C. If one module is assigned and the other is not:
 1. Determine which module is unassigned (sender or receiver)
 2. Map them:
 - a. Find the quickest data link with room for the unmapped module in its appropriate node (either sender or receiver) that contains the assigned module in its appropriate node (either sender or receiver)
 - b. Assign module

- D. Repeat A-D until all software links have been examined or until everything has been mapped
- E. Randomly map any unassigned modules.

DISTRIBUTION

	<u>Copies</u>		<u>Copies</u>
Defense Technical Information Center		University of Illinois	
Cameron Station		Attn: Dept. of Computer Science	
Alexandria, VA 22314	12	Dr. Jane W-S. Liu	1
		Dr. Kwei-Jay Lin	1
Library of Congress		1304 W. Springfield Ave.	
Attn: Gift and Exchange Division	4	Urbana, Il 61801	
Washington, DC 20540			
Naval Air Development Center		Internal Distribution:	
Attn: Code 7033		D4	1
(Dr. C. Schmiedekamp)	1	E231	2
(P. Zombori)	1	E232	3
Warminster, PA 18974-5000		E342 (GIDEP)	1
		F01	1
Office of Naval Technology		F31 (W. Laposata)	1
Attn: Code 227		G07 (F. Moore)	1
(CDR J. Van Fossen)	1	G42 (T. Dumoulin)	1
800 N. Quincy Street		G42 (A. Farsaie)	1
Arlington, VA 22217-5000		G42 (J. Moscar)	1
		G42 (E. Ogata)	1
Center for Naval Analyses		G42 (J. Youngblood)	1
4401 Fort Avenue		G70 (D. Dorsey)	1
P.O. Box 16268		G72 (H. Parks)	1
Alexandria, VA 22302-0268	2	H32 (J. Miller)	1
		K02	1
United States Army		K10 (J. Sloop)	1
CECOM, C2NVEO		K12 (J. O'Toole)	1
Attn: AMSEL/RD/VNT/TST		K13 (D. Parks)	1
(H. Nguyen)	1	K14 (D. Clark)	1
Fort Belvoir, VA 22060		K41 (L. Gross)	1
		K51 (J. Smith)	1
Advanced Technology & Research Corp.		K52 (G. Brooks)	1
Attn: Adrien J. Meskin	5	K52 (W. Farr)	1
Goerge Stathopoulos	1	K52 (H. Huber)	1
14900 Sweitzer Lane		N15 (M. Wilson)	1
Laurel, MD 20707		N35 (M. Masters)	1
		N35 (F. Riedl)	1
Computer Command and Control Company		N35 (M. Zarin)	1
Attn: Evan Lock	1	R44 (E. Cohen)	1
2300 Chestnut Street, Suite 230		R44 (H. Szu)	1
Philadelphia, PA 19103		R44 (J. Wingate)	1

R44 (J. Zien)	1
U	1
U02	1
U042	1
U10	1
U20	1
U30	1
U33	1
U302 (P. Hwang)	20
U33 (D. Choi)	1
U33 (M. Edwards)	1
U33 (K. Murphy)	1
U33 (N. Hoang)	1
U33 (S. Howell)	10
U33 (M. Jenkins)	1
U33 (T. Moore)	1
U33 (C. Nguyen)	20
U33 (T. Park)	1
U33 (H. Roth)	1
U33 (M. Trinh)	1
U33 (P. Wallenberger)	1
U40	12

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE October 1990		3. REPORT TYPE AND DATES COVERED	
4. TITLE AND SUBTITLE Expert Design Advisor				5. FUNDING NUMBERS	
6. AUTHOR(S) Steven L. Howell, Phillip Q. Hwang and Cuong M. Nguyen					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Surface Warfare Center (U33) 10901 New Hampshire Avenue Silver Spring, MD 20903-5000				8. PERFORMING ORGANIZATION REPORT NUMBER NAVSWC TR 90-46	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES					
12a. DISTRIBUTION/AVAILABILITY STATEMENT				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report presents a methodology for the development of large, complex systems, such as Anti-Submarine Warfare systems, involving next generation mission critical computing resources. This work also demonstrates that the method is automatable using an artificial intelligence, expert system shell. The methodology of this system optimizes the design structure and generates near-optimal allocations of logical design objects onto physical implementations based on heuristic classes of rules, classes of algorithms, sets of analytical equations, and sets of probabilistic reasoning.					
14. SUBJECT TERMS decision-aided toolset Mission Critical Computing Resources				15. NUMBER OF PAGES	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR		

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and its title page. Instructions for filling in each block of the form follow. It is important to *stay within the lines* to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

BLOCK 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. (If Known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in... . When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement.

Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities.
NASA - See Handbook NHB 2200.2
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - Leave blank.
DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA - Leave blank.
NTIS - Leave blank.

Block 13. Abstract. Include a brief (*Maximum 200 words*) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (*NTIS only*)

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.